PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BOARD OF PATENT APPEALS AND INTERFERENCES

In Re Application of:	)	
	)	
David H. Lin.	)	Confirmation No. 5529
	)	
Serial No.:10/823,845	)	Examiner: Ahluwalia, Navneet K.
	)	Group Art Unit: 2166
Filed: October 14, 2004	)	
	)	
For: **Method and Apparatus for Multi-Process Access to a Linked List**	)	HP Docket: 200402290-1
	)	TKHR Docket: 050849-1390
	)	

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is an appeal from the final Office Action mailed November 6, 2008, rejecting claims 1-22 of the present application.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest of the instant application is Hewlett-Packard Development Company, a Texas Limited Liability Partnership having its principal place of business in Houston, Texas.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences.

III. STATUS OF THE CLAIMS

Claims 1-22 are pending in this application. All claims were rejected by the final Office Action and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

There have been no claim amendments made after the final Office Action, and all amendments made before the final Office Action have been entered. The claim listing in section VIII. CLAIMS – APPENDIX (below) represents the present state of the claims.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Embodiments of the claimed subject matter are summarized below with reference numbers and references to the written description ("specification") and drawings. The subject matter described below appears in the original disclosure at least where indicated, and may further appear in other places within the original disclosure.

Embodiments according to independent claim 1 involve a method for retrieving data. The method comprises: locking a linked list (p. 6 lines 1-12; FIG. 1 element 5); retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered (p. 6 lines 1-12; p. 8 lines 3-13; FIG. 1 elements 5, 10, 15, 20, and 25); marking the subsequent element in the linked-list as in-use when a breakpoint is encountered (p. 6 line 12 to p. 7 line 2; FIG. 1 element 15 and FIG. 2 element 30); creating a recommencement

reference to the subsequent element (p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40); and unlocking the linked list (p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40).

Embodiments according to independent claim 5 involve a method for deleting an element from a linked list. The method comprises: determining if the element to be deleted is in-use (p. 9 lines 1-22; FIG. 6 element 140); updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use (p. 9 lines 5-12, lines 13-22; p. 9 line 28 to p. 10 line 19) and deleting the element (p. 9 lines 1-12; FIG. 6 elements 145 and 155).

Embodiments according to independent claim 7 involve an apparatus for storing and retrieving data. The apparatus comprises: processor (p. 10 lines 20-22; FIG. 9 element 300) capable of executing an instruction sequence (p. 11 lines 5-10); memory for storing an instruction sequence (p. 10 lines 20-22; p. 11 lines 5-10; FIG. 9 element 370); input unit for receiving data (p. 10 lines 20-25; FIG. 9 element 310); first output unit for providing data according to a received data request (p. 10 lines 20-25; FIG. 9 element 320); one or more ancillary output units for providing data according to a received data request; instruction sequences stored in the memory (p. 10 lines 20-25) including: data storage module (p. 11 lines 15-16; FIG. 9 element 375) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): receive data from the input unit (p. 11 lines 18-21); allocate a data element to accommodate the data (p. 11 lines 21-23); create a reference to the data element (p. 11 lines 23-25); store the reference in at least one of a header pointer and a forward pointer included in a preceding data element (p. 11 line 23 to p. 12 line 2); and store the data in the data element (p. 11 lines 25-28); data service module (p. 11 lines 15-16; FIG. 9 element 380) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): recognize a data request from the first output unit to the exclusion of all other data requests (p. 12 lines 3-6); provide data to the first output unit from a data element according to a

data element reference (p. 12 lines 5-8) and also advance the data element reference to a subsequent data element while a breakpoint is not encountered (p. 12 lines 13-26); mark a subsequent data element as in-use when a breakpoint is encountered (p. 12 line 27 to p. 13 line 3; p. 13 lines 15-18); create a recommencement reference to a subsequent data element (p. 13 lines 3-7; p. 13 line 19 to p. 14 line 3); and enable recognition of other data requests (p. 13 lines 3-7).

Embodiments according to independent claim 13 involve a computer readable medium (p. 14 lines 18-28) having imparted thereon one or more instruction sequences for storing and retrieving data. The instruction sequences comprise: data storage module (p. 11 lines 15-16; FIG. 9 element 375) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): receive data from the input unit (p. 11 lines 18-21); allocate a data element to accommodate the data (p. 11 lines 21-23); create a reference to the data element (p. 11 lines 23-25); store the reference in at least one of a header pointer and a forward pointer included in a preceding data element (p. 11 line 23 to p. 12 line 2); and store the data in the data element (p. 11 lines 25-28); data service module (p. 11 lines 15-16; FIG. 9 element 380) that, when executed by the processor, minimally causes the processor to (p. 11 lines 5-15): recognize a data request from the first output unit to the exclusion of all other data requests (p. 12 lines 3-6); provide data to the first output unit from a data element according to a data element reference (p. 12 lines 5-8) and also advance the data element reference to a subsequent data element while a breakpoint is not encountered (p. 12 lines 13-26); mark a subsequent data element as in-use when a breakpoint is encountered (p. 12 line 27 to p. 13 line 3; p. 13 lines 15-18); create a recommencement reference to a subsequent data element (p. 13 lines 3-7; p. 13 line 19 to p. 14 line 3); and enable recognition of other data requests (p. 13 lines 3-7).

Embodiments according to independent claim 19 involve an apparatus for storing and retrieving data. The apparatus comprises: means for locking a linked list (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 lines 1-12; FIG. 1 element 5) means for retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 lines 1-12; p. 8 lines 3-13; FIG. 1 elements 5, 10, 15, 20, and 25); means for marking the subsequent element in the linked-list as in-use when a breakpoint is encountered (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 12 to p. 7 line 2; FIG. 1 element 15 and FIG. 2 element 30); means for creating a recommencement reference to the subsequent element (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40); and means for unlocking the linked list (p. 10 lines 20-25; p. 11 lines 5-17; p. 6 line 22 to p. 7 line 13; and FIG. 2 element 40).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The following grounds of rejection are to be reviewed on appeal.

A. Claims 1-22 stand rejected under 35 U.S.C § 102(e) as allegedly anticipated by *Gao et al.* (U.S. 6,898,650).

VII. ARGUMENT

A. Rejection of Claims 1-22 under 35 U.S.C. §102: *Gao et al.*

1. Independent Claim 1

Appellant submits that claim 1 is not anticipated by *Gao et al.* for at least the following reasons. For a proper rejection of a claim under 35 U.S.C. §102, the cited reference must disclose, teach, or suggest all elements/features/steps of the claim at issue. *See, e.g., E.I. du Pont de Nemours & Co. v. Phillips Petroleum Co.*, 849 F.2d 1430, 7 U.S.P.Q.2d 1129 (Fed. Cir. 1988).

a. *Gao et al.* does not teach "retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered"

The final Office Action alleges (p. 5) that *Gao et al.* discloses this feature in FIG. 3 and Col. 3 lines 51-59. Appellant disagrees. Appellant assumes that the brief mention of a next pointer in a queue is understood by a person of ordinary skill in the art to implicitly teach that the next pointer is used to advance from one element to another in the queue, and that these same techniques are applicable to the claimed "linked list". Even so, the cited portion of *Gao et al.* does not disclose, teach, or suggest "advancing to a subsequent element while a breakpoint is not encountered" – nor does any other portion, since *Gao et al.* does not discuss breakpoints at all.

The Examiner has apparently misinterpreted the term "breakpoint", since the Response to Arguments section states that "the instant application defines breakpoint as ***when control is relinquished*** and the citations are in parallel with that understanding" (Office Action, p. 2). The Examiner is mistaken about the teaching in the instant application. As pointed out in p. 3 of the previous response (filed July 30, 2008), the instant application does not state that a breakpoint is when control is relinquished, but says instead that "a breakpoint definition is used to define when a first process…***is required*** to relinquish control over the linked-list so that a second process can gain access to the linked list." (Paragraph 10, emphasis added.) *Gao et al.* teaches nothing about a process being required to relinquish control over the linked list so that a second process can gain access to it, and therefore teaches nothing about "advancing to a subsequent element while a breakpoint is not encountered". Since *Gao et al.* does not disclose, teach, or suggest all elements of claim 1, the rejection should be overturned.

b. *Gao et al.* does not teach "marking the subsequent element in the linked-list as in-use when a breakpoint is encountered"

As discussed above in section VII.A.1.a, *Gao et al.* does not discuss breakpoints at all, and thus does not disclose, teach, or suggest this claimed feature. In the interests of advancing

prosecution, Appellant will nonetheless discuss the specific allegations made in the Office Action regarding this feature.

The final Office Action alleges (p. 5) that *Gao et al.* discloses this feature in Col. 3 lines 39-50. Appellant disagrees. The cited portion of *Gao et al.* teaches an in-use flag and its meaning ("in-use flag 310 indicates whether the container is being used at the current time"), but not under what conditions this flag is set to mark any particular element as in-use. In contrast, claim 1 recites marking a ***specific*** element as in use – "the subsequent element" – under a ***specific*** condition – "when a breakpoint is encountered".

In further explaining the rejection of this claimed feature, the Response to Arguments section now relies on a different portion of *Gao et al.*:

> Furthermore, in column 4 lines 36-40 and 62-67 Gao discloses the breakpoint being encountered using figure 5 A&B using elements 510 and 525. Clearly from the figure and the cited text a breakpoint is encountered when a container that is attempted to being used and is locked and set to be in the in-use state and thus is marked.
> (Office Action, p. 3.)

Appellant disagrees. First, the condition taught by this portion of *Gao et al.* – "when a container that is attempted to being used and is locked" – is not the same as the condition recited in claim 1 – "when a breakpoint is encountered". Second, the Examiner has offered no explanation whatsoever why a person of ordinary skill in the art would understand these two to be the same.

As can be readily verified, *Gao et al.* teaches specific conditions for setting the in-use flag to indicate that a container is locked – which are ***not the same*** as the conditions recited in claim 1. Specifically, the text for block 510 teaches that the ***in-use flag is set if it is not set already***. (Col. 4, lines 35-55, discussing atomic set-and-swap to test and set a field in a single instruction cycle.) A person of ordinary skill in the art understands that the use of an atomic set-and-swap instruction allows this determination and setting to occur without interruption by a second process. The Examiner has merely made a conclusory statement that the condition taught in *Gao et al.* is the same as that recited in claim 1, rather than providing any reasoning or

evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

The Examiner refers to another block, 525, as teaching marking the subsequent element as in-use "when a breakpoint is encountered" Since the text for block 525 reads "does the container has an appropriate data valid flag", it appears that the Office Action is now taking yet another position, alleging that the claimed condition "when a breakpoint is encountered" corresponds to the teaching "when the container has an appropriate data valid flag". Appellant disagrees, and submits that the state of a data valid flag is ***not the same*** as "when a breakpoint is encountered".

Once again, the Examiner has merely made a conclusory statement that the claimed condition and the condition taught in *Gao et al.* are the same, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. For at least these reasons, *Gao et al.* does not disclose, teach, or suggest all elements of claim 1, and the rejection should be overturned.

c. *Gao et al.* does not teach "creating a recommencement reference to the subsequent element"

The final Office Action alleges (p. 5) that *Gao et al.* discloses this feature in Col. 4 lines 36-49. Appellant disagrees. This portion of *Gao et al.*, which is the text corresponding to block 510 of Figure 5, was discussed above in section VII.A.1.b. As noted there, the text for block 510 teaches setting in-use flag if it is not set already, using an atomic test-and-set operation. Since this is the only action described by this portion of *Gao et al.*, Appellant can only assume that p. 5 of the Office Action is actually alleging that the claimed action "creating a recommencement reference to the subsequent element" corresponds to the teaching "setting the in-use flag". However, the Examiner has merely made a conclusory statement that the claimed action is the same as the action taught in *Gao et al.*, rather than providing any reasoning or evidentiary

foundation as to why a person of ordinary skill in the art would understand these two to be the same.

In further explaining the rejection of this claimed feature, the Response to Arguments section relies on a different portion of *Gao et al.*:

> [T]he examiner contends that Gao discloses "creating a recommencement reference to a subsquent element". This teaching in Gao is clearly found in column 2 lines 46 – 58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use ***the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to the algorithm in figure 5A & B cited bove***. The recommencement is clearly explained and showed by the pointer in the reference. (Office Action, p. 4, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, ***the algorithm does not save state*** so that the next invocation can recommence at a different point in the linked list. That is, the algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved so that the search recommences again where it stopped before.

The Examiner alleges in the Response to Arguments section that "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head; and a next

pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "creating a recommencement reference to the subsequent element" as recited in claim 1.

Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. Since *Gao et al.* does not disclose, teach, or suggest all elements of claim 1, the rejection should be overturned.

2. Independent Claim 5

Appellant submits that claim 5 is not anticipated by *Gao et al.* for at least the reason that *Gao et al.* does not disclose, teach, or suggest "updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use". For a proper rejection of a claim under 35 U.S.C. §102, the cited reference must disclose, teach, or suggest all elements/features/steps of the claim at issue. *See, e.g., E.I. du Pont de Nemours & Co. v. Phillips Petroleum Co.*, 849 F.2d 1430, 7 U.S.P.Q.2d 1129 (Fed. Cir. 1988).

The final Office Action alleges (p. 6) that *Gao et al.* discloses this feature in Table 14. Appellant disagrees. This portion of *Gao et al.* is "pseudo-code of the process for removing data from a container not currently in use" (Col. 5, lines 15-20). Appellant submits that removing data from a container not currently in use is not the same as "updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted

when the element in is in-use". The Examiner has merely made a conclusory statement that the two actions are the same, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

In further explaining the rejection of similar functional language recited by claim 1, the Response to Arguments section (p. 4) relies on a different portion of *Gao et al.*, Col. 2 lines 46 – 58, Col. 3 lines 9-16, and Col. 3 lines 56-59. Appellant submits that this portion of *Gao et al.* does not teach a "recommencement reference" or "updating a recommencement reference" either.

This Response to Argument section states that:

> [T]he examiner contends that Gao discloses "creating a recommencement reference to a subsquent element". This teaching in Gao is clearly found in column 2 lines 46 – 58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use ***the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to the algorithm in figure 5A & B cited bove***. The recommencement is clearly explained and showed by the pointer in the reference. (Office Action, p. 4, emphasis added.)

Thus, the Examiner appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, ***the algorithm does not save state*** so that the next invocation can recommence at a different point in the linked list. That is, the algorithm does not update a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most,

that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved so that the search recommences again where it stopped before.

The Examiner alleges in the Response to Arguments section that "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head; and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use" as recited in claim 5.

Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. Since *Gao et al.* does not disclose, teach, or suggest all elements of claim 5, the rejection should be overturned.

3. Independent Claims 7 and 13

Appellant submits that claims 7 and 13 are not anticipated by *Gao et al.* for at least the following reasons. For a proper rejection of a claim under 35 U.S.C. §102, the cited reference must disclose, teach, or suggest all elements/features/steps of the claim at issue. *See, e.g., E.I. du Pont de Nemours & Co. v. Phillips Petroleum Co.*, 849 F.2d 1430, 7 U.S.P.Q.2d 1129 (Fed. Cir. 1988).

a. *Gao et al.* does not teach "data storage module that, when executed by the processor, minimally causes the processor to…advance the data element reference to a subsequent data element while a breakpoint is not encountered"

In rejecting claims 7 and 13, the Office Action alleges (p. 7 and p. 10) that *Gao et al.* discloses this feature in Col. 2 lines 31-38. The portion of *Gao et al.* relied on here is nothing more than a list of conventional computer components (*e.g.* "one or more central processing units, memory, file system, *etc.*) with no mention whatsoever of any functionality much less the specific functionality of the claimed feature noted above. The rejection of claims 7 and 13 is therefore legally deficient and should be overturned for at least this reason.

Appellant notes that the functional language in claims 7 and 13 ("advance the data element reference to a subsequent data element while a breakpoint is not encountered") does share some similarities to the functions recited in claim 1 ("advancing to a subsequent element while a breakpoint is not encountered"). Therefore, although the scope of claims 7 and 13 is not coextensive with the scope of claim 1, Appellant submits that the various portions of *Gao et al.* do not disclose, teach, or suggest "advance the data element reference to a subsequent data element while a breakpoint is not encountered", as recited in claims 7 and 13, for reasons analogous to those discussed above in connection with claim 1 (section VII.A.1).

b. *Gao et al.* does not teach "data storage module that, when executed by the processor, minimally causes the processor to…mark a subsequent data element as in-use when a breakpoint is encountered"

As discussed above in section VII.A.1.a, *Gao et al.* does not discuss breakpoints at all, and thus does not disclose, teach, or suggest this claimed feature. In the interests of advancing prosecution, Appellant will nonetheless discuss the specific allegations made in the Office Action regarding this feature.

The final Office Action alleges (p. 7 and p. 10) that *Gao et al.* discloses this feature in Col. 3 lines 39-50. Appellant disagrees. The cited portion of *Gao et al.* teaches an in-use flag and its meaning ("in-use flag 310 indicates whether the container is being used at the current

time"), but not under what conditions this flag is set to mark any particular element as in-use. In contrast, claims 7 and 13 recites marking a ***specific*** element as in use – "the subsequent element" – under a ***specific*** condition – "when a breakpoint is encountered".

In further explaining the rejection of similar functional language recited by claim 1, the Response to Arguments section relies on a different portion of *Gao et al.*:

> Furthermore, in column 4 lines 36-40 and 62-67 Gao discloses the breakpoint being encountered using figure 5 A&B using elements 510 and 525. Clearly from the figure and the cited text a breakpoint is encountered when a container that is attempted to being used and is locked and set to be in the in-use state and thus is marked.
> (Office Action, p. 3.)

Appellant disagrees. First, the condition taught by this portion of *Gao et al.* – "when a container that is attempted to being used and is locked" – is not the same as the condition recited in claims 7 and 13 – "when a breakpoint is encountered". Second, the Examiner has offered no explanation whatsoever why a person of ordinary skill in the art would understand these two to be the same.

As can be readily verified, *Gao et al.* teaches specific conditions for setting the in-use flag to indicate that a container is locked – which are ***not the same*** as the conditions recited in claims 7 and 13. Specifically, the text for block 510 teaches that the ***in-use flag is set if it is not set already***. (Col. 4, lines 35-55, discussing atomic set-and-swap to test and set a field in a single instruction cycle.) A person of ordinary skill in the art understands that the use of an atomic set-and-swap instruction allows this determination and setting to occur without interruption by a second process. The Examiner has merely made a conclusory statement that the condition taught in *Gao et al.* is the same as that recited in claims 7 and 13, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

The Examiner refers to another block, 525, as teaching marking the subsequent element as in-use "when a breakpoint is encountered" Since the text for block 525 reads "does the

container has an appropriate data valid flag", it appears that the Office Action is now taking yet another position, alleging that the claimed condition "when a breakpoint is encountered" corresponds to the teaching "when the container has an appropriate data valid flag". Appellant disagrees, and submits that the state of a data valid flag is ***not the same*** as "when a breakpoint is encountered".

Once again, the Examiner has merely made a conclusory statement that the claimed condition and the condition taught in *Gao et al.* are the same, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. For at least these reasons, *Gao et al.* does not disclose, teach, or suggest all elements of claims 7 and 13, and the rejection should be overturned.

c. *Gao et al.* does not teach "data storage module that, when executed by the processor, minimally causes the processor to...create a recommencement reference to a subsequent element"

The final Office Action alleges (p. 7 and p. 10) that *Gao et al.* discloses this feature in Col. 4 lines 36-49. Appellant disagrees. This portion of *Gao et al.*, which is the text corresponding to block 510 of Figure 5, was discussed above in section VII.A.1.b. As noted there, the text for block 510 teaches setting in-use flag if it is not set already, using an atomic test-and-set operation. Since this is the only action described by this portion of *Gao et al.*, Appellant can only assume that p. 5 of the Office Action is actually alleging that the claimed action "creating a recommencement reference to the subsequent element" corresponds to the teaching "setting the in-use flag". However, the Examiner has merely made a conclusory statement that the claimed action is the same as the action taught in *Gao et al.*, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

In further explaining the rejection of similar functional language recited by claim 1, the Response to Arguments section relies on a different portion of *Gao et al.*:

> [T]he examiner contends that Gao discloses "creating a recommencement reference to a subsquent element". This teaching in Gao is clearly found in column 2 lines 46 – 58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use ***the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to the algorithm in figure 5A & B cited bove***. The recommencement is clearly explained and showed by the pointer in the reference. (Office Action, p. 4, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, ***the algorithm does not save state*** so that the next invocation can recommence at a different point in the linked list. That is, the algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved so that the search recommences again where it stopped before.

The Examiner alleges in the Response to Arguments section that "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head; and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to

the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "creating a recommencement reference to the subsequent element" as recited in claims 7 and 13.

Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. Since *Gao et al.* does not disclose, teach, or suggest all elements of claims 7 and 13, the rejection should be overturned.

4. Independent Claim 19

Appellant submits that claim 19 is not anticipated by *Gao et al.* for at least the following reasons. For a proper rejection of a claim under 35 U.S.C. §102, the cited reference must disclose, teach, or suggest all elements/features/steps of the claim at issue. *See, e.g.*, *E.I. du Pont de Nemours & Co. v. Phillips Petroleum Co.*, 849 F.2d 1430, 7 U.S.P.Q.2d 1129 (Fed. Cir. 1988).

a. *Gao et al.* does not teach "means for…advancing to a subsequent element while a breakpoint is not encountered"

The final Office Action alleges (p. 12) that *Gao et al.* discloses this feature in FIG. 3 and Col. 3 lines 51-59. Appellant disagrees. Appellant assumes that the brief mention of a next pointer in a queue is understood by a person of ordinary skill in the art to implicitly teach that the next pointer is used to advance from one element to another in the queue, and that these same techniques are applicable to the claimed "linked list". Even so, the cited portion of *Gao et al.* does not disclose, teach, or suggest "advancing to a subsequent element while a breakpoint is not encountered" – nor does any other portion, since *Gao et al.* does not discuss breakpoints at all.

The Examiner has apparently misinterpreted the term "breakpoint", since the Response to Arguments section states that "the instant application defines breakpoint as ***when control is relinquished*** and the citations are in parallel with that understanding" (Office Action, p. 2). The Examiner is mistaken about the teaching in the instant application. As pointed out in p. 3 of the previous response (filed July 30 2008), the instant application does not state that a breakpoint is when control is relinquished, but says instead that "a breakpoint definition is used to define when a first process…***is required*** to relinquish control over the linked-list so that a second process can gain access to the linked list." (Paragraph 10, emphasis added.) *Gao et al.* teaches nothing about a process being required to relinquish control over the linked list so that a second process can gain access to it, and therefore teaches nothing about "advancing to a subsequent element while a breakpoint is not encountered". Since *Gao et al.* does not disclose, teach, or suggest all elements of claim 19, the rejection should be overturned.

b. *Gao et al.* does not teach "means for…marking the subsequent element in the linked-list as in-use when a breakpoint is encountered"

As discussed above in section VII.A.1.a, *Gao et al.* does not discuss breakpoints at all, and thus does not disclose, teach, or suggest this claimed feature. In the interests of advancing prosecution, Appellant will nonetheless discuss the specific allegations made in the Office Action regarding this feature.

The final Office Action alleges (p. 12) that *Gao et al.* discloses this feature in Col. 3 lines 39-50. Appellant disagrees. The cited portion of *Gao et al.* teaches an in-use flag and its meaning ("in-use flag 310 indicates whether the container is being used at the current time"), but not under what conditions this flag is set to mark any particular element as in-use. In contrast, claim 19 recites marking a ***specific*** element as in use – "the subsequent element" – under a ***specific*** condition – "when a breakpoint is encountered".

In further explaining the rejection of similar functional language recited by claim 1, the Response to Arguments section relies on a different portion of *Gao et al.*:

> Furthermore, in column 4 lines 36-40 and 62-67 Gao discloses the breakpoint being encountered using figure 5 A&B using elements 510 and 525. Clearly from the figure and the cited text a breakpoint is encountered when a container that is attempted to being used and is locked and set to be in the in-use state and thus is marked.
> (Office Action, p. 3.)

Appellant disagrees. First, the condition taught by this portion of *Gao et al.* – "when a container that is attempted to being used and is locked" – is not the same as the condition recited in claim 19 – "when a breakpoint is encountered". Second, the Examiner has offered no explanation whatsoever why a person of ordinary skill in the art would understand these two to be the same.

As can be readily verified, *Gao et al.* teaches specific conditions for setting the in-use flag to indicate that a container is locked – which are ***not the same*** as the conditions recited in claim 19. Specifically, the text for block 510 teaches that the ***in-use flag is set if it is not set already***. (Col. 4, lines 35-55, discussing atomic set-and-swap to test and set a field in a single instruction cycle.) A person of ordinary skill in the art understands that the use of an atomic set-and-swap instruction allows this determination and setting to occur without interruption by a second process. The Examiner has merely made a conclusory statement that the condition taught in *Gao et al.* is the same as that recited in claim 19, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

The Examiner refers to another block, 525, as teaching marking the subsequent element as in-use "when a breakpoint is encountered" Since the text for block 525 reads "does the container has an appropriate data valid flag", it appears that the Office Action is now taking yet another position, alleging that the claimed condition "when a breakpoint is encountered" corresponds to the teaching "when the container has an appropriate data valid flag". Appellant

disagrees, and submits that the state of a data valid flag is ***not the same*** as "when a breakpoint is encountered".

Once again, the Examiner has merely made a conclusory statement that the claimed condition and the condition taught in *Gao et al.* are the same, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. For at least these reasons, *Gao et al.* does not disclose, teach, or suggest all elements of claim 19, and the rejection should be overturned.

c. *Gao et al.* does not teach "means for creating a recommencement reference to the subsequent element"

The final Office Action alleges (p. 12) that *Gao et al.* discloses this feature in Col. 4 lines 36-49. Appellant disagrees. This portion of *Gao et al.*, which is the text corresponding to block 510 of Figure 5, was discussed above in section VII.A.1.b. As noted there, the text for block 510 teaches setting in-use flag if it is not set already, using an atomic test-and-set operation. Since this is the only action described by this portion of *Gao et al.*, Appellant can only assume that p. 5 of the Office Action is actually alleging that the claimed action "creating a recommencement reference to the subsequent element" corresponds to the teaching "setting the in-use flag". However, the Examiner has merely made a conclusory statement that the claimed action is the same as the action taught in *Gao et al.*, rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same.

In further explaining the rejection of similar functional language recited by claim 1, the Response to Arguments section relies on a different portion of *Gao et al.*:

> [T]he examiner contends that Gao discloses "creating a recommencement reference to a subsquent element". This teaching in Gao is clearly found in column 2 lines 46 – 58, column 3 lines 9-16 and lines 56-59, when the breakpoint has been marked and the flag set as in-use ***the recommencement point is the one at breakpoint when the flag is unset or search for another container is made according to***

> ***the algorithm in figure 5A & B cited bove***. The recommencement is clearly explained and showed by the pointer in the reference. (Office Action, p. 4, emphasis added.)

The Examiner thus appears to allege that the search for an in-use container stops at a particular container, and that the search recommences at this point next time. Appellant disagrees.

First, the algorithm in Figures 5A & 5B does not search for another container. As can be readily verified by inspection of these two Figures, the entire sequence describes a single use of the container, starting with acquiring the queue read/write lock (505) and ending with releasing the queue read/write lock (550).

Second, even assuming (for the sake of argument) that this algorithm is called multiple times by another piece of code, ***the algorithm does not save state*** so that the next invocation can recommence at a different point in the linked list. That is, the algorithm does not create a reference to a recommencement point. The portions of *Gao et al.* relied on here teach, at most, that the so-called "search" to locate a not-in-use container use stops when it has found one. It does not teach that this state is saved so that the search recommences again where it stopped before.

The Examiner alleges in the Response to Arguments section that "the recommencement is clearly explained and showed by the pointer in the reference". The portions of *Gao et al.* relied on here discuss two different pointers: a next pointer 220 in the queue head; and a next pointer 325 in the container. Appellant must therefore assume that the Examiner is specifically alleging that either next pointer 220 or next pointer 325 is a recommencement point. However, *Gao et al.* merely teaches that next pointer 220 points to any one of the containers in the queue (see Col. 2 lines 45-50; Col. 3 lines 9-16) and that next pointer 325 in each container points to the next container (see Col. 3 lines 55-60). Neither of those uses of a next pointer has anything to do with "a recommencement reference", must less the specific action of "creating a recommencement reference to the subsequent element" as recited in claim 19.

Once again, the Examiner has merely made a conclusory statement that one of these next pointers is the same as a "recommencement reference", rather than providing any reasoning or evidentiary foundation as to why a person of ordinary skill in the art would understand these two to be the same. Since *Gao et al.* does not disclose, teach, or suggest all elements of claim 19, the rejection should be overturned.

5. Dependent Claims 2-6, 8-12, 14-18, and 20-22

Since independent claims 1, 5, 7, 13, and 19 are allowable, Appellant submits that claims 2-6, 8-12, 14-18, and 20-22 are allowable for at least the reason that each depends from an allowable claim. *In re Fine*, 837 F.2d 1071, 5 U.S.P.Q. 2d 1596, 1598 (Fed. Cir. 1988). Therefore, Appellant requests that the rejection of claims 2-6, 8-12, 14-18, and 20-22 be overturned. .

B. Conclusion

For at least the reasons discussed above, Appellant respectfully requests that the Examiner's final rejection of claims 1-22 be overturned by the Board. In addition to the claims listed in Section VIII (CLAIMS – APPENDIX), Section IX (EVIDENCE – APPENDIX) included herein indicates that there is no additional evidence relied upon by this brief. Section X (RELATED PROCEEDINGS – APPENDIX) included herein indicates that there are no related proceedings.

Respectfully submitted,

By: /Karen G. Hazzah/

Karen G. Hazzah,
Reg. No. 48,472

THOMAS, KAYDEN, HORSTEMEYER & RISLEY, L.L.P.

600 Galleria Parkway, SE
Suite 1500
Atlanta, Georgia 30339-5948
Tel: (770) 933-9500
Fax: (770) 951-0933

VIII. CLAIMS – APPENDIX

1. A method for retrieving data comprising:

locking a linked list;

retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered;

marking the subsequent element in the linked-list as in-use when a breakpoint is encountered;

creating a recommencement reference to the subsequent element; and

unlocking the linked list.

2. The method of claim 1 further comprising:

locking the linked list;

determining a subsequent element in the linked list according to the recommencement reference; and

retrieving data from the determined subsequent element.

3. The method of claim 1 wherein creating a recommencement reference to the subsequent element comprises:

retrieving a pointer to the subsequent element;

determining a process identifier for a current process; and

associating the pointer with the process identifier.

4. The method of claim 1 wherein marking the subsequent element in the linked-list as in-use comprises maintaining a count of the quantity of processes that require additional access to the element.

5. A method for deleting an element from a linked list comprising:

determining if the element to be deleted is in-use;

updating a recommencement reference to the element to refer to a data element that is subsequent to the data element to be deleted when the element in is in-use; and

deleting the element.

6. The method of claim 5 wherein updating a recommencement reference to the element comprises:

discovering a pointer associated with a process identifier;

disassociating the process identifier from the pointer;

determining a pointer to a subsequent element; and

associating the process identifier with the newly determined pointer.

7. An apparatus for storing and retrieving data comprising:

processor capable of executing an instruction sequence;

memory for storing an instruction sequence;

input unit for receiving data;

first output unit for providing data according to a received data request;

one or more ancillary output units for providing data according to a received data request;

instruction sequences stored in the memory including:

data storage module that, when executed by the processor, minimally causes the processor to:

receive data from the input unit;

allocate a data element to accommodate the data;

create a reference to the data element;

store the reference in at least one of a header pointer and a forward pointer included in a preceding data element; and

store the data in the data element;

data service module that, when executed by the processor, minimally causes the processor to:

recognize a data request from the first output unit to the exclusion of all other data requests;

provide data to the first output unit from a data element according to a data element reference and also advance the data element reference to a subsequent data element while a breakpoint is not encountered;

mark a subsequent data element as in-use when a breakpoint is encountered;

create a recommencement reference to a subsequent data element; and

enable recognition of other data requests.

8. The apparatus of claim 7 wherein the data service module, when executed by the processor, further minimally causes the processor to:

recognize a data request from the first output unit to the exclusion of all other data requests; and

provide data to the first output unit from a data element according to the recommencement reference.

9. The apparatus of claim 7 wherein the data service module causes the processor to create a recommencement reference by minimally causing the processor to:

retrieve a pointer to a data element subsequent to a current data element;

determine an identifier associated with the data request received from the first output unit; and

store the retrieved pointer and the determined identifier in an associative manner.

10. The apparatus of claim 7 wherein the data service module causes the processor to mark a subsequent data element as in-use by minimally causing the processor to increment a use counter included in a subsequent data element.

11. The apparatus of claim 7 wherein the data service module further minimally causes the processor to receive a delete data request from an output unit by minimally causing the processor to:

determine if a data element to be deleted is in-use;

update a recommencement reference to refer to a data element that is subsequent to the data element to be deleted; and

delete the data element according to the received delete data request.

12. The apparatus of claim 11 wherein the data service module causes the processor to update a recommencement reference by minimally causing the processor to:

discover a pointer according to a data request identifier; and

replace the pointer with a pointer to a data element that is subsequent to the data element to be deleted.

13. A computer readable medium having imparted thereon one or more instruction sequences for storing and retrieving data comprising:

data storage module that, when executed by a processor, minimally causes the processor to:

receive data from an input unit;

allocate a data element to accommodate the data;

create a reference to the data element;

store the reference in at least one of a header pointer and a forward pointer included in a preceding data element; and

store the data in the data element;

data service module that, when executed by a processor, minimally causes the processor to:

recognize a data request from a first output unit to the exclusion of all other data requests;

provide data to a first output unit from a data element according to a data element reference and also advance the data element reference to a subsequent data element while a breakpoint is not encountered;

mark a subsequent data element as in-use when a breakpoint is encountered;

create a recommencement reference to a subsequent data element; and

enable recognition of other data requests.

14. The computer readable medium of claim 13 wherein the data service module, when executed by a processor, further minimally causes the processor to:

recognize a data request from a first output unit to the exclusion of all other data requests; and

provide data to a first output unit from a data element according to the recommencement reference.

15. The computer readable medium of claim 13 wherein the data service module causes a processor to create a recommencement reference by minimally causing the processor to:

retrieve a pointer to a data element subsequent to a current data element;

determine an identifier associated with a data request received from a first output unit; and

store the retrieved pointer and the determined identifier in an associative manner.

16. The computer readable medium of claim 13 wherein the data service module causes a processor to mark a subsequent data element as in-use by minimally causing the processor to increment a use counter included in a subsequent data element.

17. The computer readable medium of claim 13 wherein the data service module further minimally causes the processor to receive a delete data request from an output unit by minimally causing the processor to:

determine if a data element to be deleted is in-use;

update a recommencement reference to refer to a data element that is subsequent to the data element to be deleted; and

delete the data element according to the received delete data request.

18. The computer readable medium of claim 17 wherein the data service module causes the processor to update a recommencement reference by minimally causing the processor to:

discover a pointer according to a data request identifier; and

replace the pointer with a pointer to a data element that is subsequent to the data element to be deleted.

19. An apparatus for storing and retrieving data comprising:

means for locking a linked list;

means for retrieving data from an element in the linked list and also advancing to a subsequent element while a breakpoint is not encountered;

means for marking the subsequent element in the linked-list as in-use when a breakpoint is encountered;

means for creating a recommencement reference to the subsequent element; and

means for unlocking the linked list.

20. The apparatus of claim 19 further comprising:

means for locking the linked list;

means for determining a subsequent element in the linked list according to the recommencement reference; and

means for retrieving data from the determined subsequent element.

21. The apparatus of claim 19 further comprising a means for deleting an element in the linked-list.

22. The apparatus of claim 21 wherein the means for deleting an element comprises:

means for determining if the element to be deleted is in-use;

means for updating a reference to the element to refer to a subsequent element in the linked list when the element in is in-use; and

means for deleting the element.

IX. EVIDENCE – APPENDIX

None.

X. RELATED PROCEEDINGS – APPENDIX

None.